UNIVERSIDADE REGIONAL DO NOROESTE DO ESTADO DO RIO GRANDE DO SUL

LUIZ GABRIEL GRUMICKER PEREIRA
PEDRO ARTHUR DE OLIVEIRA BUTZKE

Pré-Apresentação Teórica: BuBank

Programação Orientada a Objetos

Santa Rosa
2024

1 INTRODUÇÃO

Este trabalho contém todas as informações necessárias para o entendimento do projeto de uma corretora digital chamada BuBank, desenvolvida para oferecer serviços bancários e de investimento. O sistema permite que os usuários realizem operações como transferências, saques e investimentos em produtos financeiros, incluindo LCI, LCA, CDI e outros.

2. OBJETIVO

O principal objetivo do BuBank é permitir que os usuários gerenciem suas finanças de forma acessível, oferecendo ferramentas para que possam administrar suas contas e investir em diferentes produtos financeiros. O projeto visa simplificar o controle financeiro doméstico, se usado como um projeto pessoal, e como um exemplo de banco caso mais amplamente aplicado.

3. FONTE DE DADOS A SER UTILIZADA

Para simulação de operações e cálculo de rendimentos, são utilizados dados hipotéticos de produtos financeiros, como taxas de rendimento associadas a LCI, LCA e CDI. No futuro, dados reais poderiam ser integrados a partir de APIs de corretoras ou bancos, permitindo atualizações em tempo real das cotações dos produtos financeiros.

4. METODOLOGIA

O desenvolvimento do BuBank utiliza Java como linguagem de programação principal, implementando conceitos de Programação Orientada a Objetos (POO) para criar uma arquitetura modular e fácil de manter. Como ferramentas, será utilizada a IDE NetBeans para o desenvolvimento. Além disso, frameworks estão sendo pensados ainda, conforme o desenvolvimento prático da aplicação.

5. DESENVOLVIMENTO

O sistema bancário do Bubank vai contar com algumas classes principais, como Banco, Pessoa, Usuário, Conta, derivando Conta Corrente e Conta de Investimentos, além de classes de apoio, como nós nomeamos, que será as classes Transação e Produto, essas classes serão utilizadas como “tipo” para outra classe, além de deixar o código muito mais organizado posteriormente. Algumas classes principais também foram utilizadas como “tipo” em outros atributos de outras classes, mas aquelas eram de certa forma mais importantes que essas de apoio.

5. 1 CLASSE BANCO

A classe Banco representa o banco "Bubank" e é responsável por gerenciar contas e usuários associados. Ela possui o atributo **nome**, que armazena o nome do banco, o atributo contas, uma lista das contas associadas ao banco, e o atributo **usuários**, que armazena a lista de usuários com contas no banco. O atributo **qtdContas** registra o total de contas no banco, sendo armazenado como um número inteiro.

O construtor **Banco(String nome)** inicia o banco com o nome especificado e listas vazias para as contas e os usuários. A classe também oferece diversos métodos para gerenciamento: o método **adicionar Conta** adiciona uma conta ao banco e registra o usuário associado à conta; o método **remover Conta** remove uma conta e, se aplicável, o usuário, atualizando o contador de contas. Para localizar uma conta, o método **buscar Conta(String numero, String agencia)** permite buscar uma conta específica com base em seu número e agência. Já o método login(String username, String senha) verifica as credenciais fornecidas e retorna o usuário correspondente caso as credenciais estejam corretas.

Para recuperar informações sobre o banco, o método **listar Contas()** retorna todas as contas do banco, enquanto **getQtdContas()** retorna a quantidade total de contas. Por fim, os métodos **getNome()** e **setNome(String nome**) permitem acessar e atualizar o nome do banco, respectivamente.

5. 2 CLASSE PESSOA

Pessoa representa uma pessoa física, incluindo informações de contato e endereço. Entre seus atributos principais, temos **nome**, armazenando o nome da pessoa como uma **String**, e **cpf**, que representa o CPF da pessoa, definido como constante e imutável após a criação. A data de nascimento é armazenada no atributo **dataNascimento**, do tipo **Date**. Para contato, a classe possui os atributos **email**, para o endereço de e-mail, e **telefone**, que registra o número de telefone, ambos como strings.

Além dos dados de contato, a classe também gerencia informações de endereço com os atributos **endereco**, que contém o endereço residencial completo, **cidade**, **estado** e **cep**, todos representados por strings, que detalham a localização da pessoa.

A classe conta com um construtor, **Pessoa**(...), que inicializa todos esses atributos. Ela oferece métodos de acesso e modificação, como **getNome**() e **setNome** para obter e definir o nome, e **getCpf**(), que retorna o CPF da pessoa.. Métodos como **getDataNascimento**() e **setDataNascimento** permitem acessar e atualizar a data de nascimento, enquanto **getEmail** e

setEmail, e **getTelefone** e **setTelefone** permitem gerenciar o e-mail e telefone, respectivamente. Para o endereço, **getEndereco**() e **setEndereco** acessam e definem o endereço residencial, enquanto **getCidade**(), **getEstado**(), e **getCep**() (com seu correspondente método **set**) permitem o gerenciamento de cidade, estado e CEP. Essa estrutura fornece uma maneira abrangente e flexível de gerenciar dados de uma pessoa.

5. 3 CLASSE USUÁRIO

A classe **Usuário** contém informações de autenticação e status, além de herdar dados básicos de uma pessoa. Seus atributos incluem uma **senha** (do tipo String), usada para acesso ao sistema, um **username** para login, um indicador **ativo** para mostrar se o usuário está ativo ou inativo, um atributo **admin** que indica privilégios de administrador e, por fim, a **dataCadastro**, que registra a data de criação do usuário como uma constante.

O construtor **Usuário** inicia o usuário com uma senha, um nome de usuário e dados de ‘‘**Pessoa”**, configurando-o como ativo e não-administrador por padrão. Os métodos principais incluem **getSenha** e **setSenha** para obter e modificar a senha, **getUsername** e **setUsername** para manipular o nome de usuário, **isAtivo** e **setAtivo** para verificar ou alterar o status ativo do usuário, **isAdmin** e **setAdmin** para verificar ou definir privilégios de administrador, e **getDataCadastro**, que retorna a data de cadastro do usuário.

5. 4 CLASSE CONTA

A classe **Conta** representa uma conta bancária de um usuário. Ela possui três atributos principais: **usuário**, que representa o usuário proprietário da conta; **numeroConta**, que armazena o número da conta gerado aleatoriamente como um número de 4 dígitos; e **agencia**, que inicialmente é configurada como "0001". O construtor **Conta(Usuario)** cria uma nova conta para o usuário fornecido, gerando automaticamente um número de conta aleatório e configurando a agência para o valor padrão. A classe oferece métodos para acessar e modificar esses atributos: o método **getNumeroConta()** retorna o número da conta; **getAgencia()** retorna o número da agência; e **setAgencia(String agencia)** permite definir um novo número de agência. O método **getUsuario()** retorna o usuário associado à conta.

5. 4. 1 CLASSE CONTA CORRENTE

Representando uma conta corrente, essa classe permite ao usuário realizar operações bancárias essenciais, como depósitos, saques, e transferências, além de fornecer um histórico das transações realizadas. Esse histórico permite ao usuário visualizar todas as atividades financeiras associadas à conta.

Entre os principais atributos da classe, temos o **saldo**, que armazena o valor disponível na conta corrente em um formato **double**, o atributo **conta**, que representa a conta básica associada a essa conta corrente, sendo do tipo **Conta**, e o **extrato**, uma lista de transações realizadas, usada para gerar o histórico detalhado da conta.

O construtor **Conta Corrente** inicializa uma nova conta corrente com saldo zero e uma lista vazia para as transações, criando assim um ponto de partida para o registro de todas as operações. Com o método **depositar**, o usuário pode realizar depósitos na conta, que aumentam o saldo disponível, e essa transação é registrada automaticamente no extrato. O método **sacar** permite a realização de saques, deduzindo o valor do saldo caso haja fundos suficientes e registrando a transação no histórico.

Para obter informações sobre o saldo e as transações, o método **getSaldo** retorna o saldo atual da conta corrente, enquanto **getExtrato** fornece a lista completa das transações registradas. A classe também oferece dois métodos de transferência: o método **transferir**, que permite transferir fundos para uma conta de investimento, e o método **transferir**, que realiza transferências para outra conta corrente, ambos registrando a transação no extrato e descontando o valor do saldo. Por fim, o método **getConta** retorna a conta básica associada à conta corrente, permitindo o acesso às informações da conta original.

5. 4. 2 CLASSE CONTA DE INVESTIMENTOS

A classe **Conta Investimento** permite ao usuário realizar depósitos, saques, transferências para conta corrente e acessar um extrato das transações realizadas. Os principais atributos são o **saldo**, que armazena o valor disponível, uma **conta** associada do tipo **Conta**, o **extrato**, uma lista de transações, e por fim a **Lista de Produtos** que é uma lista de todos os produtos (ações, LCA, LCI, CDI, CDB, Cripto) que o usuário possui. O construtor inicializa o saldo como zero e o extrato como vazio.

Os métodos incluem **depositar** e **sacar** para adicionar ou remover saldo com registro no extrato, **getSaldo** para consultar o saldo atual, **getExtrato** para retornar a lista de transações, e **transferir** para enviar fundos a uma **Conta Corrente**. Existe também o método **getConta**, que retorna a conta associada.

Esta classe deve ser responsável por armazenar o saldo alocado para investimentos e gerenciar as operações de compra e venda de ativos (Produtos).

5. 5 CLASSE TRANSAÇÃO - TIPO

A classe **Transação** tem por objetivo representar uma transação bancária executada em uma conta, armazenando informações como tipo, valor, saldo após a operação, data e uma descrição opcional da transação. Este tipo de dado será utilizado para compor o extrato das classes Conta Corrente e Conta de Investimentos, permitindo o registro detalhado de cada movimentação realizada.

Os atributos que caracterizam a classe incluem tipo, que será nesse caso um texto (String) que indica o tipo da transação (como depósito, saque ou transferência), um valor, do tipo double, que armazena o valor monetário envolvido, um saldoRestante, também double, que guarda o saldo da conta após a transação, a data, do tipo Date para registrar a data e o horário em que ocorreu a transação, e por fim a descricao, uma variável de texto que possibilita uma descrição opcional da transação.

Entre os métodos, o construtor com todos os atributos da classe, exceto descrição e data que será colocado valores padrão para descrição até uma edição futura e data atual para não possibilitar modificar isso. Os métodos de acesso (getters) são organizados em um conjunto que permite a obtenção dos valores de cada atributo: **getTipo**(), **getValor**(), **getData**(), **getSaldoRestante**() e **getDescricao**(). O método **setDescricao**(String descricao) possibilita a atualização ou inclusão de uma descrição para a transação. Além disso, **getTransacao**() retorna uma string formatada com todos os detalhes da transação, incluindo tipo, valor, saldo restante, data e descrição.

5. 6 CLASSE PRODUTO - TIPO

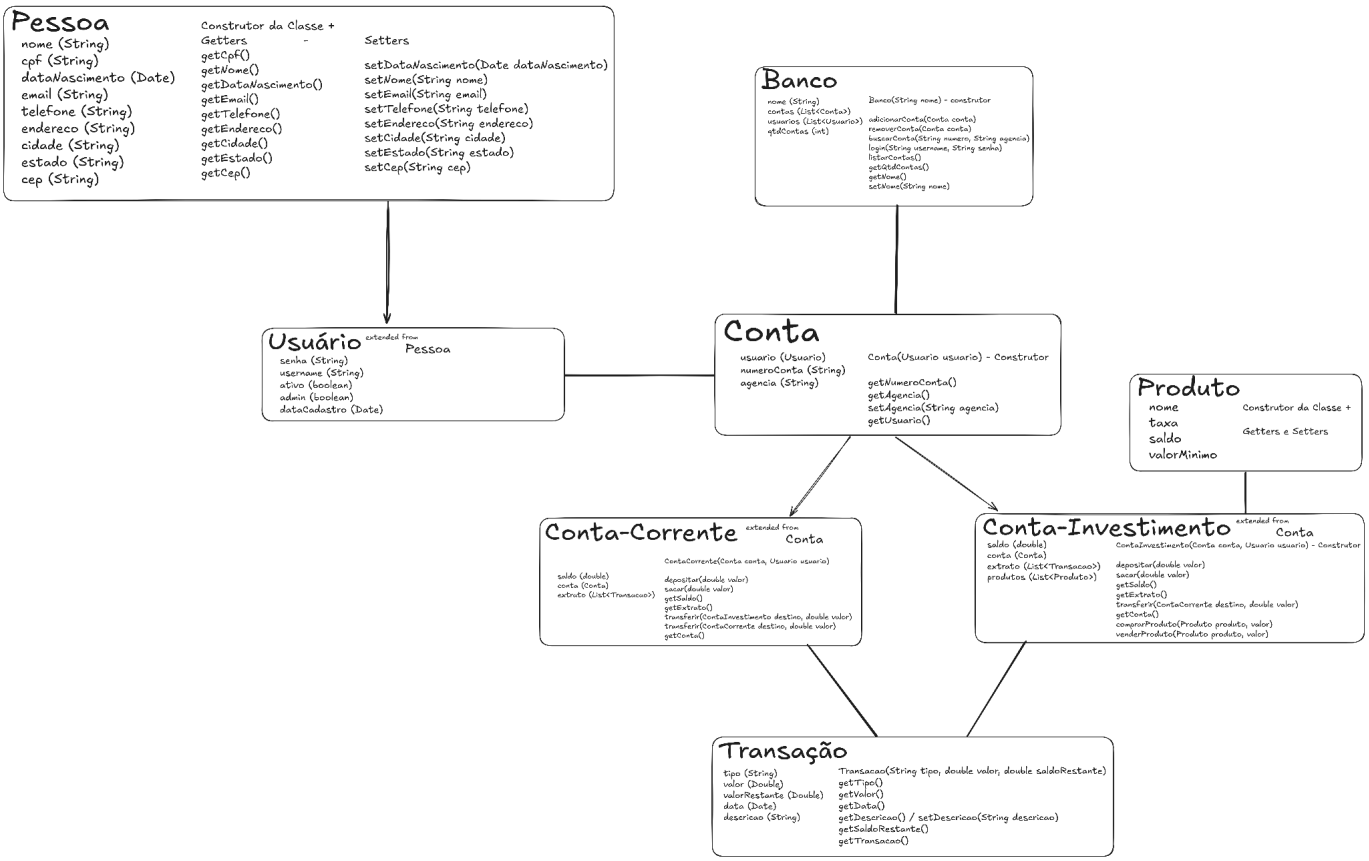
A classe Produto foi desenvolvida para representar produtos de investimento financeiro, como LCI, LCA e CDI, que podem ser adquiridos e vendidos por contas de investimento. Ela possibilita a adição e remoção desses produtos em uma lista dentro da classe Conta Investimento, facilitando a organização e o controle dos investimentos realizados pelos clientes.

A Classe possui atributos como nome, do tipo texto (String), taxa (double), para representar a taxa de rendimento do produto, saldo (double) que representa o valor investido, o valor mínimo (double) que representa o valor mínimo, seja de uma ação, CDB ou Cripto.

Essa classe também conta com um construtor que é possível inicializar a classe com os dados de todos os atributos, e também conta com métodos para obter e atualizar os atributos da ação.

6 DIAGRAMA

Abaixo mostramos um diagrama desenvolvido por nós para tudo que comentamos nesse documento:



Fonte: Os autores

7 CONCLUSÃO

A elaboração deste trabalho nos deu uma visão mais clara sobre como criar e desenvolver um sistema bancário digital, como o BuBank. Ao analisarmos as classes e suas funções, ficou claro como é importante ter um código bem estruturado para que o sistema funcione bem e seja eficiente.

Agora, estamos prontos para a fase prática, onde vamos colocar em ação tudo o que aprendemos, desenvolvendo as funcionalidades que planejamos. Este projeto não apenas nos ajudou a entender melhor a teoria, mas também estimulou nossa criatividade e vontade de trazer novas ideias e soluções para um sistema bancário.

8 FEEDBACK DA APRESENTAÇÃO

- Atualizar a Classe Transação para implementar uma rastreabilidade, colocando a Origem e Destino, além de colocar o Produto para uma rastreabilidade de investimentos (como é usada pelos dois, provável que sempre um vai ficar vazio)
- Remover o isAtivo e isAdmin por não comentar o que realmente esses atributos vão modificar no nosso sistema, e para deixar mais simples.